AI-Powered Drug Toxicity Prediction Pipeline

Context Overview

We are building an end-to-end toxicity prediction pipeline to evaluate candidate molecules across general, organ-specific, and specialized safety dimensions. This system integrates advanced ML/DL models (transformers, GNNs, XGBoost), RDKit featurization, and explainability layers (SHAP, Captum), all orchestrated through Ray, Celery, and FastAPI. Core sources are `toxicity_classifier` and `Convexia_demo`.

Directory Structure

```
toxicity-pipeline/
├── README.md
├── pyproject.toml              # poetry; Python 3.10
├── Makefile                    # dev targets: lint, format, test
├── .env.example
├── docker/
│   ├── api.Dockerfile
│   ├── model-gpu.Dockerfile
│   └── model-cpu.Dockerfile
├── toxicity/
│   ├── config/                 # default.yaml, scoring weights
│   ├── data/                   # PAINS, BRENK, etc.
│   ├── descriptors/            # RDKit, MACCS, ECFP
│   ├── models/
│   │   ├── base.py             # ToxicityModel ABC
│   │   ├── general/            # DeepTox, TDC2, XGBoost
│   │   ├── organ/              # hERG, DILI, Kidney, H-optimus-0,
UNI, Merlin
│   │   ├── specialised/        # Neuro (CONVERGE), MITO (MITO-Tox),
Immuno (TxGemma)
│   │   ├── adme/               # CYP450, pkCSM, enzyme risk
```

```
|   |   └── structural/         # PAINS, RM alerts
|   ├── pipeline/               # ray executor, aggregator
|   ├── explain/                # SHAP, Captum, SVG visualization
|   ├── api/                    # FastAPI
|   ├── cli/                    # predict.py, batch.py
|   ├── db/                     # SQLAlchemy models, migrations
|   ├── tasks/                  # Celery workers
|   ├── utils/                  # logging, chemistry utilities
|   └── tests/                  # unit, integration, load
└── docs/
    └── architecture.md
```

Core Implementation Steps

Step 1: Merge Repositories & Normalize

- Extract deep learning models from `toxicity_classifier`
- Extract CLI, Docker, logging, and XGBoost models from `Convexia_demo`
- Organize models under `toxicity/models/<category>`
- Enforce code formatting using `black` and `ruff`
- Add continuous integration using `make test` and `make format`

Step 2: Define Model Interface

```
class ToxicityModel(ABC):
    name: str
    version: str
    timeout_s: int

    @abstractmethod
    def predict(self, smiles: str) -> dict:
        return {
            "score": float,
            "confidence": float,
```

```
            "raw_output": Any,
            "metadata": dict,
        }
```

- All models (e.g., DeepTox, hERG) will inherit from this interface
- Include RDKit parsing internally and raise errors for invalid inputs
- Add a default `batch_predict()` method

Step 3: Add Config Loader

```
@dataclass
class ModelCfg:
    enabled: bool
    path: Path
    gpu: bool = False
    timeout: int = 30

settings = Settings.parse_yaml("toxicity/config/default.yaml")
```

- Supports enabling/disabling models and assigning scoring weights per use-case

---

Step 4: Input Preprocessing

- Validate SMILES strings using RDKit
- Compute:
    - ECFP6 (2048-bit)
    - MACCS keys (166-bit)
    - Molecular descriptors (MW, LogP, TPSA, etc.)
    - 3D conformer generation (for future use)
- Cache results to optimize batch inference

Model Layers

General Toxicity

- **TDC2 transformer model** (HuggingFace, fine-tuned on Tox21)
- **DeepTox** transformer model (wrap with output parser)
- **XGBoost** baseline (from `Convexia_demo`)
- **Output**: 0–1 normalized score

Organ-Specific Models

- **hERG**: Add IC_{50} regression head and similarity-based confidence
- **DILI (Liver)**: Combine XGBoost with structural alerts
- **Nephrotoxicity**: GNN model with renal filter heuristics
- **H-optimus-0**: For rare pattern recognition
- **Merlin / UNI (optional)**: CT-based and image-driven predictions
- **Output**: Per-organ risk

Metabolism

- **TxGemma**:
  - Wrap enzyme interaction predictors (CYP450, Phase I/II)
  - Extract CYP450 interaction profiles and associated enzyme-level risks
- **pkCSM** (fallback): Extract metabolism-related predictions
- **Output**: CYP interaction + enzyme risk level

Neurotoxicity

- **CONVERGE-inspired model**:
  - logBB > 0.3, MW < 450, logP 1–4, <5 H-bond donors
  - ToxCast-based prediction
- **Output**: 0–1 normalized neurotoxicity score

Mitochondrial Toxicity

- **MITO-Tox**:

    - Predict ATP depletion, membrane disruption
    - Integrate with known mitochondrial stress pathways
- **Output**: Mitochondrial risk probability

Immunotoxicity

- **TxGemma-style model**:
    - Predicts cytokine release, e.g., IFN-γ, IL-6
    - Use XGBoost trained on immune-response data
- **Output**: Immunotoxicity risk level

Tissue Accumulation

- **pkCSM**:
    - Predict per-organ accumulation
    - Penalize excessive volume of distribution
- **Output**: Per-organ accumulation → converted to penalty score

---

Aggregation Layer

Scoring Formula

```
final_score = (
  0.15 * general_tox +
  0.20 * organ_tox_avg +
  0.15 * neurotox +
  0.10 * mito_tox +
  0.10 * morpho_tox +
  0.10 * accumulation_penalty +
  0.10 * immunotox +
  0.10 * structural_alert_penalty
)
```

- Normalize all scores to a 0–1 scale
- Profile-specific weight configurations (e.g., discovery vs. preclinical)

Confidence Estimation

- Based on variance between models
- Highlight disagreements between model categories

Explainability

- Tree-based: **SHAP**
- Deep learning: **Captum** (IntegratedGradients)
- Molecule heatmaps using RDKit + SVG
- Log disagreement and justification for low-confidence outputs

---

API and CLI Interface

API (FastAPI)

```
@app.post("/predict")
async def predict(req: schemas.PredictionIn):
    job_id = tasks.run_pipeline.delay(req.smiles, req.profile)
    return {"job_id": job_id}
```

- `/results/{job_id}` to poll prediction status
- Celery for async job handling across CPU/GPU

CLI

```
tox-cli predict "CC(=O)OC1=CC=CC=C1C(=O)O"
tox-cli batch input.smi --out results.json
```

Database Schema

```
compounds (
  id UUID,
  smiles TEXT,
  canonical_smiles TEXT,
  created_at TIMESTAMP
)

predictions (
  compound_id UUID,
  model_name TEXT,
  model_version TEXT,
  prediction JSONB,
  confidence FLOAT,
  created_at TIMESTAMP
)
```

- Predictions cached for 90 days
- Cache evicted on model version changes
- Total cache capped at 10GB (LRU)

Testing Plan

Unit Tests

- Individual models, CLI, featurization logic
- Error handling, expected outputs

Integration Tests

- End-to-end prediction flow
- Invalid SMILES, long inference, GPU fallback

Performance Benchmarks

- ≤30 sec for single molecule
- ≤10 min for batch of 100
- 50 simultaneous users supported
- RAM <16GB, GPU <4GB

Model Validation

- High-toxicity compounds → high scores
- Safe, FDA-approved drugs → low scores
- Tox21/ToxCast alignment within ±10%

Dependencies (Poetry)

```
[tool.poetry.dependencies]
python = "^3.10"
rdkit-pypi = "^2022.9"
torch = "^2.2"
transformers = "^4.42"
xgboost = "^2.0"
scikit-learn = "^1.5"
pydantic = "^2.8"
fastapi = "^0.111"
uvicorn = { extras=["standard"], version="^0.30" }
ray = "^2.24"
celery = "^5.4"
redis = "^5.0"
SQLAlchemy = "^2.0"
asyncpg = "^0.29"
alembic = "^1.13"
shap = "^0.45"
captum = "^0.7"
pyyaml = "^6.0"
```

Docker Services

```
services:
  api:
    build: docker/api.Dockerfile
    ports: ["8000:8000"]
  model-gpu:
    build: docker/model-gpu.Dockerfile
    runtime: nvidia
  model-cpu:
    build: docker/model-cpu.Dockerfile
    scale: 2
  redis:
    image: redis:6-alpine
  postgres:
    image: postgres:13
```

📦 Data & Model Governance

Data Pipeline & Versioning

Problem: The current spec does not describe how datasets (e.g. Tox21, ToxCast, hERG DBs, in-house) are ingested, cleaned, or versioned.

Solution:

- Implement a reproducible, modular data ingestion pipeline (`toxicity/data/etl/`) with the following stages:
    - **Ingestion:** Raw dataset parsing and schema enforcement (CSV, SDF, JSON).
    - **Cleaning:** Removal of duplicates, invalid SMILES, and standardization using RDKit.
    - **Feature Extraction:** Descriptor and fingerprint generation.
    - **Splitting:** Time-split or scaffold-split for train/val/test sets.

Tooling:

- Use **DVC** to version datasets, intermediate files, and model artifacts.
- Store large files via **Git LFS** or cloud-backed DVC remotes (e.g. S3).

Structure:

```
toxicity/data/
├── raw/
├── processed/
├── splits/
└── dvc.yaml
```

Experiment Tracking & Model Registry

Problem: There's no formal experiment tracking or registry to manage model metrics, lineage, or deployment status.

Solution:

- Integrate **MLflow** (or Weights & Biases) with the training pipeline:
  - Track hyperparameters, datasets, git commit, training run UUID.
  - Log metrics (AUC, F1, calibration), model artifacts, and plots.
  - Tag runs by model type, endpoint, and task (e.g. `toxicity/hERG`).

Model Registry:

- Use MLflow Model Registry to store production-ready models.
- Define stage transitions: `Staging → Production → Archived`.

CI/CD for Code & Models

Code CI/CD:

- Add to `toxicity/.github/workflows/ci.yaml`:
  - Linting: `ruff`, `black`
  - Unit + integration tests: `pytest`
  - Docker build verification
  - FastAPI route tests (basic smoke tests)

Model CI/CD:

- Auto-retraining trigger:

    - On new commit to `models/**`, `data/**`, or `config/**`
    - Launch retraining + validation pipeline
    - Log model to MLflow + version in DVC
- Add Slack/GitHub notifications on model promotion or failure

Bonus:

- Build `toxicity/registry.py` to retrieve latest `production` model via MLflow or config loader
- Periodically revalidate top-performing models on held-out test sets